COMS 4774 Spring 2021 Scribe: Daniel Hsu (djh2164)
September 1, 2017 Editor: Chris Alberti (ca2790)

Batch learning via online prediction

1 Introduction

One of the major applications of online learning is batch learning (i.e., offline learning). In batch learning for binary classification, there is a sequence of labeled examples

$$\big((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\big) \in (\mathcal{X} \times \{\pm 1\})^n$$

called the "training data", and the goal is to find a classifier with small prediction error. The training data are assumed to be drawn i.i.d. from a fixed probability distribution $\mathbb{P}$. The prediction error of a classifier $f \colon \mathcal{X} \to \{\pm 1\}$ is the probability that the classifier mis-classifies a random example (x, y) drawn from $\mathbb{P}$:

$$\mathrm{err}(f) := \mathbb{P}(f(x) \neq y).$$

A batch learning algorithm takes as input the training data and returns a classifier. We'll also talk about randomized classifiers f; the probability defining $\mathrm{err}(f)$ is then taken with respect to both the internal randomization in f as well as the random draw (x, y) from $\mathbb{P}$.

We abstractly think of an online prediction algorithm as taking as input a sequence of n labeled examples, and producing a sequence of classifiers $f_1, f_2, \ldots, f_n, f_{n+1} \colon \mathcal{X} \to \{\pm 1\}$. The t-th classifier f_t can only depend on the first $t-1$ labeled examples. Although not all online prediction algorithms fit this template, it is sufficient for most cases.

What algorithms don't fit this template?

2 Expected prediction error vs mistake bounds

In expectation, the prediction error of the classifier returned by the batch learning algorithm $\mathcal{B}$ can be written as

$$\mathbb{E}\left(\mathbb{1}_{\{\mathcal{B}(S_{1:n})(x_{n+1}) \neq y_{n+1}\}}\right)$$

where

$$S_{1:n+1} := \big((x_1, y_1), (x_2, y_2), \ldots, (x_{n+1}, y_{n+1})\big)$$

is a sequence of $n+1$ labeled examples drawn i.i.d. from $\mathbb{P}$, and $S_{1:n}$ denotes the sequence of the first n such examples. If $\mathcal{B}$ is a randomized algorithm, then the expectation above is also taken with respect to the random bits used by $\mathcal{B}$.

The following is a simple way to convert an online prediction algorithm $\mathcal{A}$ such as Perceptron and Winnow into a (randomized) batch learning algorithm $\mathrm{OTB}_{\mathcal{A}}$ in a way that bounds the expected prediction error of $\mathrm{OTB}_{\mathcal{A}}$ in terms of a mistake bound for $\mathcal{A}$.

Algorithm 1 Online-to-batch learning algorithm $\mathrm{OTB}_{\mathcal{A}}$

input Training data $S_{1:n} := ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$.

1: Run $\mathcal{A}$ with the sequence $S_{1:n}$ to produce classifiers $f_1, f_2, \ldots, f_n, f_{n+1}$.
2: Pick $T \in \{1, 2, \ldots, n+1\}$ uniformly at random.
3: **return** f_T.

Another way to interpret Algorithm 1 is as follows.

1. Run $\mathcal{A}$ on the training data sequence, and keep all $n+1$ classifiers $f_1, f_2, \ldots, f_n, f_{n+1}$. generated along the way.

2. The final classifier used for prediction is a randomized classifier: to predict on a new instance x, first randomly choose $f_T \in \{f_1, f_2, \ldots, f_{n+1}\}$, and then predict $f_T(x)$.

Theorem 1. *Let $M_{\mathcal{A},n+1}$ be the number of mistakes made by online prediction algorithm $\mathcal{A}$ on a sequence of $n+1$ i.i.d. examples $S_{1:n+1}$ drawn from $\mathbb{P}$. The expected prediction error of $\mathrm{OTB}_{\mathcal{A}}(S_{1:n})$ is $\mathbb{E}(M_{\mathcal{A},n+1})/(n+1)$.*

Proof. By direct computation:

$$\begin{aligned}
\mathbb{E}\left(\mathbb{1}_{\{\mathrm{OTB}_{\mathcal{A}}(S_{1:n})(x_{n+1}) \neq y_{n+1}\}}\right) &= \sum_{t=1}^{n+1} \Pr(T=t) \cdot \mathbb{E}\left(\mathbb{1}_{\{\mathrm{OTB}_{\mathcal{A}}(S_{1:n})(x_{n+1}) \neq y_{n+1}\}} \,|\, T=t\right) \\
&= \sum_{t=1}^{n+1} \frac{1}{n+1} \mathbb{E}\left(\mathbb{1}_{\{f_t(x_{n+1}) \neq y_{n+1}\}}\right) \\
&= \sum_{t=1}^{n+1} \frac{1}{n+1} \mathbb{E}\left(\mathbb{1}_{\{f_t(x_t) \neq y_t\}}\right) \\
&= \frac{1}{n+1} \mathbb{E}\left(\sum_{t=1}^{n+1} \mathbb{1}_{\{f_t(x_t) \neq y_t\}}\right).
\end{aligned}$$

The penultimate step is true because f_t is only a function of $S_{1:t-1}$, and because the distribution of $(S_{1:t-1}, (x_{n+1}, y_{n+1}))$ is the same as that of $(S_{1:t-1}, (x_t, y_t))$. The sum in the final expectation is $M_{\mathcal{A},n+1}$. □

Theorem 1 shows that an online prediction algorithm with a small mistake bound can be effectively used for batch learning. If the mistake bound of $\mathcal{A}$ for a sequence of $n+1$ examples is sublinear in n, then the expected prediction error of $\mathrm{OTB}_{\mathcal{A}}$ is $o(1)$. Indeed, the mistake bound for Perceptron and Winnow are constant in the "realizable" setting (when there is a half-space function that is always correct with a margin). In the "agnostic" setting (i.e., when no such perfect half-space function is assumed to exist), it possible to show that the expected error of $\mathrm{OTB}_{\mathcal{A}}$ is at most the optimal hinge-loss achieved by a half-space function specified by norm-bounded weight vectors, plus a term that is roughly $O(1/\sqrt{n})$.

Computationally, Algorithm 1 can be quite attractive if the online prediction algorithm has constant time updates and has a constant memory requirements (with respect to n). Perceptron and Winnow certainly fit this bill. In these cases, the running time scales linearly with n. In fact, the algorithm only needs streaming access to the data.

One drawback of Algorithm 1 is that its final classifier is a randomized classifier: it randomly picks among the $n+1$ classifiers generated by the online prediction algorithm, and uses the chosen classifier to predict. However, the randomized classifier can be *derandomized* by using a majority vote over the $n+1$ classifiers. The prediction error of the majority-vote classifier is at most twice the prediction error of the randomized classifier.

Some online prediction algorithms such as Perceptron and Winnow don't update their weight vectors unless a mistake is made, and thus there may actually be fewer than $n+1$ different classifiers. In such cases, it can be more efficient to represent the final randomized (or majority-vote) classifier as a weighted combination of these different classifiers. The weight is proportional to the number of rounds in which it is used by the online prediction algorithm—specifically, one plus the number of examples on which it predicts correctly.

In the case of half-space functions, it is reasonable to use a (weighted) average of the weight vectors corresponding to the different classifiers. This can be thought of as an approximation to the majority-vote classifier. In some cases, averaging can be rigorously justified and in fact shown to at least as good as the randomized classifier.

3 Progressive validation

Another significant advantage of using online prediction algorithms for batch learning is that one can estimate the prediction error of the learned classifier very accurately, without the need for a separate held-out test set. This technique is called *progressive validation*.

To motivate progressive validation, we first review how validation is often performed in batch learning. Suppose n labeled examples $S_{1:n} := ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$ are available. These examples are split into two groups: the first $n - m$ examples $S_{1:n-m}$ and the final m examples $S_{n-m+1:n}$. The first $n - m$ examples $S_{1:n-m}$ are given to a batch learning algorithm $\mathcal{B}$ to learn a classifier $\hat{f} := \mathcal{B}(S_{1:n-m})$. The last m examples $S_{n-m+1:n}$ are used to compute the *test error* of $\hat{f}$:

$$\widehat{\mathrm{err}}_{\mathrm{test}}(\hat{f}) := \frac{1}{m} \sum_{t=n-m+1}^{m} \mathbb{1}_{\{\hat{f}(x_t) \neq y_t\}}.$$

Under the assumption that the original n examples are drawn i.i.d. from some fixed distribution $\mathbb{P}$, the test error $\widehat{\mathrm{err}}_{\mathrm{test}}(\hat{f})$ is an unbiased estimator of the prediction error of $\hat{f}$:

$$\mathbb{E}\left(\widehat{\mathrm{err}}_{\mathrm{test}}(\hat{f})\right) = \mathrm{err}(\hat{f}).$$

Note that this is not true of the *training error* of $\hat{f}$:

$$\widehat{\mathrm{err}}_{\mathrm{train}}(\hat{f}) := \frac{1}{n-m} \sum_{t=1}^{n-m} \mathbb{1}_{\{\hat{f}(x_t) \neq y_t\}}.$$

Indeed, $\hat{f}$ is generally not independent of $S_{1:n-m}$ because those examples are used by the batch learning algorithm to produce $\hat{f}$. However, the last m examples $S_{n-m+1:n}$ are independent of $\hat{f}$, so

$$\mathbb{E}\left(\mathbb{1}_{\{\hat{f}(x_t) \neq y_t\}}\right) = \mathrm{err}(\hat{f})$$

for $n - m + 1 \leq t \leq n$, and hence $\mathbb{E}(\widehat{\mathrm{err}}_{\mathrm{test}}(\hat{f})) = \mathrm{err}(\hat{f})$. The following proposition is a standard application of standard binomial tail bounds (e.g., Hoeffding's inequality). It can be improved considerably, although we shall not discuss it here because the same improvements will also apply to the approach we discuss next.

Proposition 1. *Let $\hat{f} := \mathcal{B}(S_{1:n-m})$ for any batch learning algorithm $\mathcal{B}$. For any $\delta \in (0, 1)$,*

$$\Pr\left(\mathrm{err}(\hat{f}) \leq \widehat{\mathrm{err}}_{\mathrm{test}}(\hat{f}) + \sqrt{\frac{\ln(1/\delta)}{2m}}\right) \geq 1 - \delta$$

where the probability is taken with respect to $S_{n-m+1:n}$.

Progressive validation provides a way to (essentially) use *all n data* for both training and testing, as long as the training is done using the online-to-batch procedure $\mathrm{OTB}_{\mathcal{A}}$ with some online prediction algorithm $\mathcal{A}$. Let $\hat{f} := \mathrm{OTB}_{\mathcal{A}}(S_{1:n-1})$ be the final randomized classifier produced by $\mathrm{OTB}_{\mathcal{A}}$ using the first $n - 1$ examples. That is, $\hat{f}$

is the randomized classifier that picks one of $f_1, f_2, \ldots, f_n$ uniformly at random and uses the chosen classifier to predict. The estimator for the prediction error of $\hat{f}$ is

$$\widehat{\mathrm{err}}_{\mathrm{pv}}(\hat{f}) := \frac{1}{n} \sum_{t=1}^{n} \mathbb{1}_{\{f_t(x_t) \neq y_t\}}$$

i.e., the number of mistakes made by $\mathcal{A}$ on $S_{1:n}$, divided by n.

Proposition 2. *Let $\hat{f} := \mathrm{OTB}_{\mathcal{A}}(S_{1:n-1})$. For any $\delta \in (0,1)$,*

$$\Pr\left(\mathrm{err}(\hat{f}) \leq \widehat{\mathrm{err}}_{\mathrm{pv}}(\hat{f}) + \sqrt{\frac{\ln(1/\delta)}{2n}}\right) \geq 1 - \delta$$

where the probability is taken with respect to $S_{1:n}$.

Proof. For each $t = 1, 2, \ldots, n$, define $Z_t := \mathrm{err}(f_t) - \mathbb{1}_{\{f_t(x_t) \neq y_t\}}$. Since f_t only depends on $S_{1:t-1}$, it follows that $\mathbb{E}(Z_t \,|\, S_{1:t-1}) = 0$. Furthermore, $Z_t \leq 1$. Therefore $Z_1, Z_2, \ldots, Z_n$ is a martingale difference sequence. By the Azuma-Hoeffding inequality, for any $\epsilon > 0$

$$\Pr\left(\frac{1}{n}\sum_{t=1}^{n} Z_t > \frac{\epsilon}{n}\right) \leq \exp\left(-2\epsilon^2/n\right).$$

The probability bound is equal to δ when $\epsilon := \sqrt{n \ln(1/\delta)/2}$. Therefore, with probability at least $1 - \delta$,

$$\frac{1}{n}\sum_{t=1}^{n} \mathrm{err}(f_t) - \widehat{\mathrm{err}}_{\mathrm{pv}}(\hat{f}) = \frac{1}{n}\sum_{t=1}^{n} \left(\mathrm{err}(f_t) - \mathbb{1}_{\{f_t(x_t) \neq y_t\}}\right) \leq \sqrt{\frac{\ln(1/\delta)}{2n}}.$$

The conclusion follows because the randomized classifier $\hat{f}$ has prediction error $\mathrm{err}(\hat{f}) = \sum_{t=1}^{n} \mathrm{err}(f_t)/n$. □

Thus, while Proposition 2 lacks the generality of Proposition 1, it compensates by allowing the training procedure $\mathrm{OTB}_{\mathcal{A}}$ to use $n - 1$ examples and providing an estimator of the prediction error that deviates by no more than $O(1/\sqrt{n})$ with high probability.

Bibliographic notes

The online-to-batch conversion for expected prediction error is due to Helmbold and Warmuth [3], and the deterministic (majority-vote and averaging) variants were proposed for use with Perceptron by Freund and Schapire [2]. The progressive validation technique was proposed by Blum et al. [1].

References

[1] A. Blum, A. Kalai, and J. Langford. Beating the hold-out: Bounds for K-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 203–208, 1999.

[2] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[3] D. P. Helmbold and M. K. Warmuth. On weak learning. *Journal of Computer and System Sciences*, 50:551–573, 1995.